ELEC40006: 1st Year Electronics Design Project 2021

Initial Specification

Ed Stott and Esther Perea

May 2021

1. Introduction

The purpose of this module is to bring together the knowledge and skills you have gained throughout this academic year by applying them to a practical problem.

You will also acquire new skills that will help you complete the project successfully.

The project is conducted in groups of three.

2. Timeline

Week no.	Date	Action item
2	5th May	Project Brief released, introduction meeting
	9th May	Project selection and group formation deadline
3	11th May	Confirmation of group composition
9	13th June	Report, video, code and source files submission

Choose your group and project option by completing this spreadsheet: Year 1 Project 2021 Group Selection.xlsx

3. Deliverables

Report (50%): The report is a formal documentation of all the technical and non-technical work you have done on the project. By this time all your design decisions will have been made and you will be able to document the performance of various aspects of your system / simulation. You should also have a clear plan for any work outstanding before you can complete the demonstration. One team member should act as overall editor to ensure that the report is consistent.

Video demonstration (50%): The team will record a video explaining their final design, alongside a demonstration.

Code and source files for plagiarism checking

Rules for sharing

1. You **may** discuss the project with students from other groups
2. You **may** share ideas and try out others' suggestions
3. You **may** share links to useful resources
4. You **may not** share your own designs or data (schematic files, code, screenshots, plots)
5. You **may not** share elements of your report or demo video

4. Project Options

Choose one of the following options by Sunday 9th May:

Option 1: Analogue Music Synthesiser

Analogue music production remains popular today, despite the availability of digital alternatives, thanks to its authenticity. In this option, you will design a circuit for an analogue music synthesiser and simulate it using LTSpice.

Essential Requirements

a. The input to the synthesiser shall be a voltage that represents the frequency.
b. The synthesiser shall generate audio frequency tones in the range 261.6Hz – 493.9Hz (the C_4 octave), with frequency related to input voltage by $\mathrm{f} = 55\mathrm{V}$.
    i. (*Advanced*) Generate tones for a larger range of notes, up to 27.5Hz – 4.186kHz, the range of an 88-key piano. In this case use a logarithmic input voltage function, with $\mathrm{f} = 2^{V+4.781}$
c. All the components in your simulation must be real components (passive components, or semiconductors with part numbers) and not 'behavioural' models, with the following exceptions:
    i. DC voltage sources may be used as power supplies.
    ii. A voltage source in PWL mode may be used to generate the keyboard input to the synthesiser, representing a sequence of key presses.
    iii. Additional voltage sources in PWL mode may be used to represent user inputs (such as control knobs) to any enhanced functions that you design.
    iv. Any behavioural component may be used to test a sub-circuit before it is integrated into the final design.
d. The synthesiser must drive a loudspeaker with 8Ω impedance

Optional Requirements

You can also add any functionality that enhances your design. Possibilities include:

a. Choice of waveforms/instruments
b. Modulation, including tremolo (amplitude modulation) and vibrato (frequency modulation)
c. Enveloping (another form of amplitude modulation)
d. Filtering (usually with a transfer function that varies with time or a user input)
e. Low frequency oscillator to drive the modulation and filter blocks
f. Arpeggiator or other sequencing

You'll need to conduct your own research to decide what functionality should be included and how it should be implemented. Existing circuits can be adapted if they are correctly referenced, and the report shows that their operation is understood, and their use is appropriate.

Use the MATLAB script supplied on Blackboard to listen to your outputs and visualise them with a spectrogram plot. You should adopt a modular and systematic approach to break the system down into sub-circuits, and develop each sub-circuit independently before you integrate the overall system.

Evaluation

1. The quality of a music synthesiser is a subjective criterion, but try comparing the sound it makes with recorded samples of commercial synthesisers that you find on the internet

2. Calculate the BOM (bill of materials) cost for your design by summing up the costs of all the components you have used
3. Find the power consumption of your design and investigate if it varies as you use different functions. Find how the power consumption breaks down between the different circuit blocks.

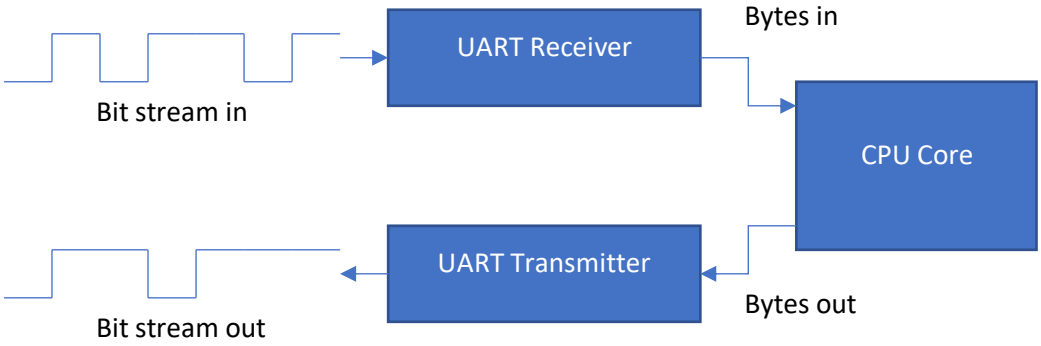
Option 2: Central Processing Unit

An efficient CPU implementation is optimised to solve commonly occurring computing problems. Features must be chosen carefully to achieve the best performance in the greatest number of applications for the smallest number of transistors. Starting with your MU0-ARM CPU design from the DECA labs, implement and integrate the three extra features described below.

Serial communication

Serial communication is used for many types of input and output in computers such as human interface devices, network communications and storage. One of the simplest forms of serial communication is UART (universal asynchronous receiver transmitter).

Serial communication is usually implemented in a CPU by dedicated logic. The UART logic handles the communication bit-by-bit so the CPU only needs to respond once entire bytes have finished transmission.



UART logic is often implemented as a *memory-mapped* device, where certain memory addresses access special registers in the device logic instead of locations in RAM. For a UART, there might be one address to write a byte for transmission, one address to read a received byte, and one address to read the state of the UART block (e.g. has a byte been received since the last byte was read?). A memory map for a CPU might look like this:

Address	**Target**
0x000 … 0x7ff	RAM
0x800	UART Status
0x801	UART Receive
0x802	UART Transmit
0x803 … 0xfff	Unused

Create a UART block with the following specifications:

1. The UART protocol shall be 1 start bit (logic 0), 8 data bits (LSB first) and 1 stop bit (logic 1). The receive and transmit signals should be logic 1 between transmissions.
2. The bit rate shall be 1 bit per 4 clock cycles
3. The CPU shall be able to read the status of the UART block to determine:
    I. If a byte has been received since the receive register was last read by the CPU
    II. If a transmission is in progress
    III. If there was a receive overflow (a byte was received before the previous byte was read by the CPU)
    IV. If there was a transmit overflow (a byte was written by the CPU before the previous byte had finished transmission)

Use your UART block to receive bytes from a test stimulus block and, after each one, transmit the accumulated sum of all the bytes that have been received so far.

Advanced: Hardware blocks like a UART are usually supported by interrupts. An interrupt works like an automatic function call that is triggered by an event. When the UART finishes sending or receiving a byte, the interrupt is triggered to prompt the CPU to copy the next byte for transmission or process the byte that was just received.

An interrupt can occur at any time and cause the program counter to jump to a specific location. After an interrupt happens, *context saving* is required so that the CPU can be restored to its original state when the interrupt function finishes. That way, the function of the main program will not be affected by interrupt. For example, if the interrupt function will modify register 0, it must first save register 0 to memory and then copy it back later in case its contents is required by the main program.

Test interrupts by placing the summation and transmit code in the interrupt function and running code for one of the other tasks in the background to show that context saving works correctly.

Floating point arithmetic

Computer algorithms that work with real-world numbers and measurements often must support a larger *dynamic range* than is possible with binary integers. For example, a circuit simulation might have to calculate currents anywhere between 1nA and 1kA, and to represent all those values as integers with a resolution of 0.1% would require a variable that could store up to 10^{15}pA (50 bits).

The solution is to use a *floating-point* representation, which represents numbers as a significand and exponent, like a scientific decimal notation $a \times 10^b$. Floating point representations allow more accurate processing of data but require more complicated logic to carry out arithmetic operations.

Use dedicated logic and/or program functions to implement floating point operations with the following specifications:

1. The operations add, subtract and multiply shall be supported
2. The number format shall be 16-bit IEEE 754 *half-precision*, with 1 sign bit, 11 significand bits and 5 exponent bits
    a. (Advanced) Use 32-bit IEEE 754 single precision.
3. All numbers except zero shall be treated as *normal*, which means that the MSB of the mantissa/significand is assumed to be 1 and is not stored. Zero is a special case represented by setting all the exponent bits to 0 and ignoring the mantissa bits.
    a. (Advanced) Add support for *denormal/subnormal* numbers, which are non-zero numbers with the minimum possible exponent and a mantissa MSB of 0.

The rounding behaviour of operations is not defined in this specification; rounding up or rounding down is acceptable and it doesn't need to be consistent. The result of exceptions (e.g. divide by zero, overflow) is also undefined.

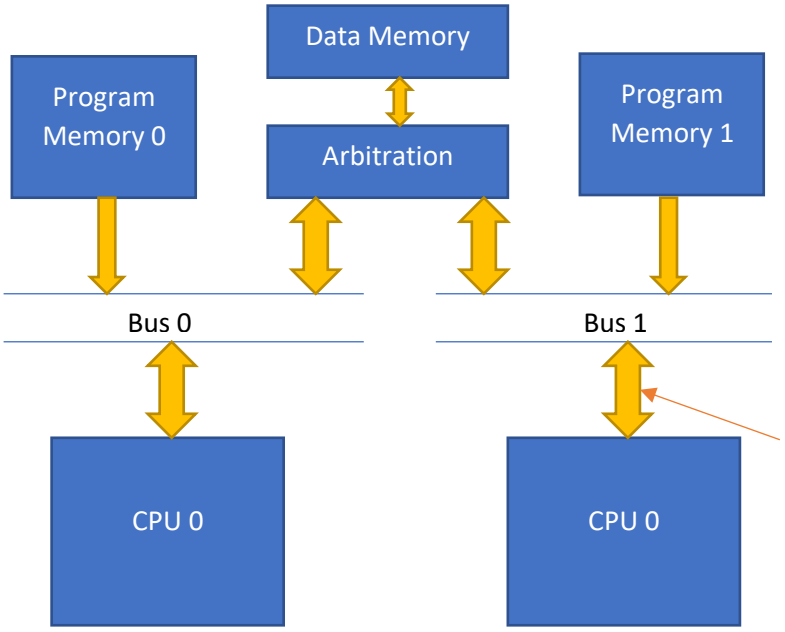
Test your CPU by implementing a function to calculate $\sin x$ using a Maclaurin series for values $-\pi \leq x \leq \pi$ and different orders of the series expansion. Compare the accuracy of the results with a fixed-point implementation. You can store values of $\frac{1}{n!}$ in memory to avoid using a division operation.

Dual core CPU

Many modern processors have multiple cores and they can execute more than one execution *thread* simultaneously. Each CPU core has its own program counter and set of CPU registers, but there is just one main memory so that cores can access the same data.

When multiple CPU cores access the same memory, *bus arbitration* logic is needed to handle events when both CPUs try to access the memory at the same time. A CPU will need to stall if the memory is busy when it tries to access data.

Your Mu0-ARM CPU accesses the memory to read every instruction as well as data. If two CPUs tried to retrieve instructions from the same memory the resulting memory bottleneck would remove the performance benefit of implementing two cores. Therefore, give each CPU its own instruction memory and share only the data memory.



Some extra signals are added to support bus arbitration. Reads and writes are triggered by request signals (replacing WEN). Read Valid and Write Complete are used to signal to the CPU when the read or write operation has been completed.

Implement a dual core CPU and use it to calculate the mean of an array of 2^n numbers. Accelerate the calculation by allocating half of the array to each CPU. Find the acceleration compared to a single CPU core, relative to the size of the array.

Evaluation

Each feature has an evaluation method for checking its correctness and performance. In addition, carry out the following evaluations:

1. Find the speed of the CPU by counting the number of clock cycles required to run the benchmarks and find how this figure changes with the size of the problem.
2. Use the FPGA compilation tool to find the maximum clock frequency of your design (see extra instructions) so you can covert clock cycle counts into execution times.
3. The power consumed by a digital circuit relates approximately to the number of logic gates and the clock speed. Find the number of logic gates (see extra instructions) to estimate the overall power consumption.

Option 3: Circuit Simulator

Write a software package that performs a small-signal AC analysis simulation of a circuit, like LTspice. The main elements of such a system are described below

Parse the netlist file

The netlist should be described in a file using a reduced SPICE format, which will be provided. You will need to read the file, convert the circuit to a small signal equivalent and store it in a suitable data structure.

Set up the simulation

An AC analysis takes place by finding the transfer function of the linearised (small signal equivalent) circuit for different frequencies over a specified range. The circuit must have a voltage or current source that is defined as the input, and a nominated node for the output. Reactive components are replaced by a complex impedance according to their value and the frequency step. Any non-linear components are replaced by their small-signal equivalent circuits. Non-dependent sources are replaced by open or short circuits.

Construct and solve the conductance matrix

You have seen how nodal analysis can be performed by writing an equation for each node that satisfies Kirchoff's current law, then solving these equations simultaneously to find the unknown node voltages. Such systems of linear equations can be solved systematically by writing them in matrix form and solving for the vector of unknowns.

$$\begin{bmatrix} G_{11} & -G_{12} & \dots & -G_{1n} \\ -G_{21} & G_{22} & \cdots & -G_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -G_{n1} & -G_{n2} & \dots & G_{nn} \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \end{bmatrix}$$

Equation for finding a vector of unknown node voltages from a conductance matrix

Here, G_{12} is the conductance directly connecting nodes 1 and 2, G_{11} is the total conductance connected to node 1, i_1 is the total current from constant current sources entering the node 1, and v_1 is the unknown voltage of node 1. The reference node is not included in the conductance matrix and instead its voltage is defined as zero.

The solution is found by calculating the inverse of the conductance matrix. This is complex but common operation in computing, so it makes sense to use a library, but you must justify your selection of a library and how you have used it in your report. Remember that your voltages, currents and conductances may be complex numbers.

Voltage and Current sources

Voltage and current sources are removed from the circuit for AC analysis, except for the input source and any dependent sources.

Voltage sources must be treated specially since the conductance of an ideal voltage source is infinite and so it cannot appear in the conductance matrix. If one terminal of the voltage source is connected to the reference node then the node connected to the other terminal can be expressed simply as something like $v_1 = v_{src}$, e.g. for a source v_{src} volts connected to node 1. The conductances connected to that node are ignored and in matrix form that would look like this:

$$\begin{bmatrix} 1 & 0 & \dots & 0 \\ -G_{21} & G_{22} & \cdots & -G_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -G_{n1} & -G_{n2} & \dots & G_{nn} \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} v_{\text{src}} \\ i_2 \\ \vdots \\ i_n \end{bmatrix}$$

Circuit analysis equation containing a voltage source between node 1 and reference

If the voltage source is connected between two non-reference nodes then you need to consider the unknown current through the voltage source, i_{VSRC}. On paper, you might do this by applying KCL to a supernode that envelops the voltage source. In linear algebra it is simpler to add i_{VSRC} to the vector of unknowns so that you can still write a KCL equation for each node:

$$\begin{bmatrix} 1 & -1 & \cdots & 0 & 0 \\ G_{11} & 0 & \cdots & -G_{1n} & -1 \\ 0 & G_{22} & \cdots & -G_{2n} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -G_{n1} & -G_{n2} & \cdots & G_{nn} & 0 \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \\ i_{\text{VSRC}} \end{bmatrix} = \begin{bmatrix} v_{\text{src}} \\ i_1 \\ i_2 \\ \cdots \\ i_n \end{bmatrix}$$

Circuit analysis equation containing a voltage source between node 1 and node 2

In this example there is a voltage source connected between nodes 1 and 2. The conductance between nodes 1 and 2 is set to zero because it cannot be calculated. Instead, an extra column is added to account for the current that flows through the voltage source.

The only dependent source that you need to include is a voltage-dependent current source, which is relatively simple to add to your circuit equation by adding the transconductance to the relevant matrix cells. For example, if a node is connected to a current source with a function $i = (v_1 - v_2)g_m$ then you would add g_m to column 1 and subtract it from column 2.

Write the output

The output of the analysis is the transfer function between the input source and a nominated output node. The output is written to a file describing the frequency of each step that was calculated with the magnitude and phase of the transfer function at that frequency. Write the output in CSV format with each frequency step as a row. Use the MATLAB script provided on blackboard to plot the results.

DC Operating Point (advanced)

The AC analysis is based on a linear circuit, where non-linear devices are replaced by small-signal equivalents. However, the small signal models are calculated using a DC operating point, which does require the solution of the full device equations for non-linear components. To get started, you can use assumptions and annotations (constants) to specify the operating point, for example you could assume that V_{be} of a BJT is 0.7V.

LT Spice and other simulators calculate the operating point by performing an initial solution for the DC circuit. There is no analytic solution for circuits containing non-linear components in the general case, so instead a numerical solution is calculated using the iterative Newton Raphson method.

The component is converted to a linear approximation (a Thevenin or Norton equivalent) by guessing the node voltages and differentiating the I-V characteristic to obtain its gradient. Then, nodal analysis is performed as usual. The results of the nodal analysis are used to refine the node voltages, and from that derive a new linear approximation. The process is repeated until the node voltages stop changing between each iteration and converge on the solution.

Evaluation

Your solution should be evaluated against the following criteria:

1. Accuracy: compare the outputs to pen and paper solutions. For simple circuits you should be able to calculate an exact solution for comparison. You can also compare with LTspice.
2. Efficiency: find how long the simulation takes and, by estimating the power consumption of your computer, the amount of energy needed. How does it scale with the number of nodes in the circuit? Are there any implementation choices that affect the efficiency?